BENDER 

QUICK DESIGN

USER MANUAL

Copy Right Notice

17 Cardinale Lane
Queensbury, NY 12804

Phone: (518) 743-8755
Fax: (518) 743-8755
E-mail: rbender@BenderRBT.com
Web Site: www.BenderRBT.com

Technical Support

If you have any questions about this manual or this software, contact the Bender RBT Inc. support group. The hours are Monday-Friday, 8:00 AM to 6:00 PM PST. The fax number and e-mail options are available 24 hours a day, 7 days a week.

- E-mail: support@BenderRBT.com
- Phone: (707) 538-1932
- Fax: (707) 538-1481

Software License Notice

Your license agreement with Bender RBT Inc. authorizes the proper use and duplication of the BenderRBT software. Any unauthorized duplication of the software in whole or in part, in print, or in any other storage and retrieval system is forbidden.

Trademarks

BenderRBT, Quick Design, Caliber-RBT, and CaliberRBT are trademarks of Bender RBT Inc. All other product and company names are trademarks or registered trademarks of their respective owners.

Table of Contents

1. Introduction

There are two challenges in designing a good set of tests for software:

1. You need to minimize the number of tests while still providing strong coverage

2. You need to ensure that you are getting the right answer for the right reason.

For even a relatively simple system the number of possible test suites actually exceeds the number of molecules in the universe (which is 10^{80} according to Stephen Hawking in "A Brief History In Time"). The key challenge then is to select an infinitesimally small subset of tests which, if they run correctly, give you a very high degree of assurance that all of the other combinations/permutations will also run correctly.

The issue in ensuring that you got the right answer for the right reason involves the fact that two or more defects may cancel each other out under some circumstances. You get the right answer for the wrong reason. To solve this, tests must "sensitized" to ensure any defects will be seen at an observable point.

Quick Design, as with all other combinatorics based test design engines, addresses only reducing the number of tests to a manageable level. The Cause-Effect Graphing test design engine portion of RBT addresses both test optimization and observability of defects since it sensitizes the test paths to ensure that any logic defect will propagate to an observable point. If the functions you are testing are business critical, mission critical, and/or safety critical, we recommend using the Cause-Effect Graph based test case design. Quick Design should be used for non-critical functions or an initial shakedown of critical functions – which would then be followed by C-E Graph based tests. Another area where Quick Design is appropriate is in designing configuration tests and creating seed tests for performance testing.

Quick Design is based on pair wise testing algorithms. In this approach you have multiple variables. Each variable has one or more states. Pairs are created by the cross of each variable/state with each other variable/state done two variable/states at a time. These pairs are then merged into tests in such a way that each pair is included (i.e. covered) in at least one test. If the mapping is done via orthogonal pairs, then each pair occurs in the same number of tests. This generally adds to the number of tests in the final test suite.

What Quick Design adds to traditional pairs wise testing is the inclusion of constraints. Applying constraints across the variables/states ensures that you do not produce combinations that would be impossible to create in real life. If constraints are not factored into the test design process, the tester must spend a lot of time editing and adjusting the tests prior to implementing them.

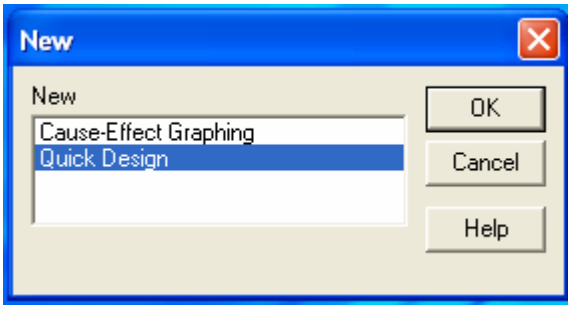
If you are not familiar with pairs wise testing and such concepts as equivalence class testing with boundary analysis, don't worry. As we go through each section on using the Quick Design portion of the RBT tool, the underlying process will be explained. Of course that means you actually have to read the User Manual, but it is a small price to pay.

2. Entering Quick Design

BenderRBT includes two test design engines. If you double click on the BenderRBT icon

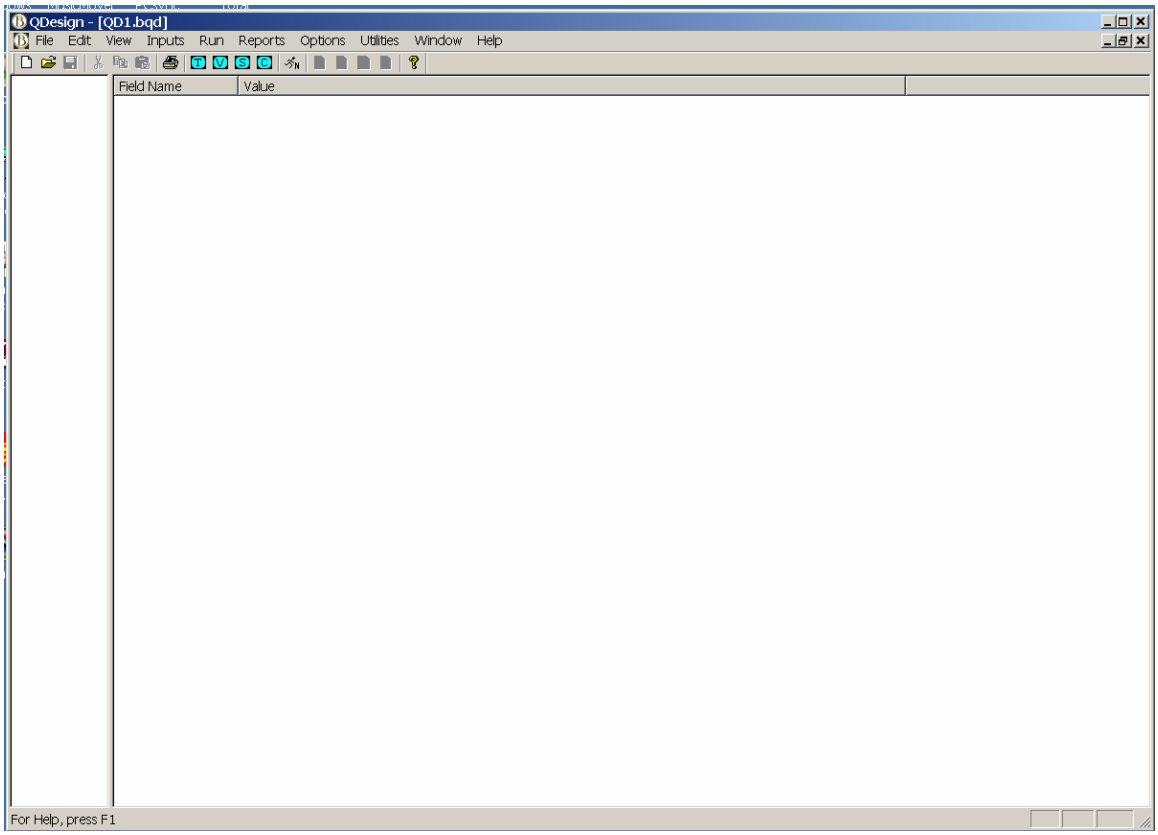
RBT Icon

you will be presented with a choice of Cause-Effect Graphing or Quick Design.



RBT Test Design Engine Options

Cause-Effect Graphing takes you to the main screen of the Graphing based test design engine. To enter Quick Design select it and hit OK. That will take you to the main Quick Design screen:

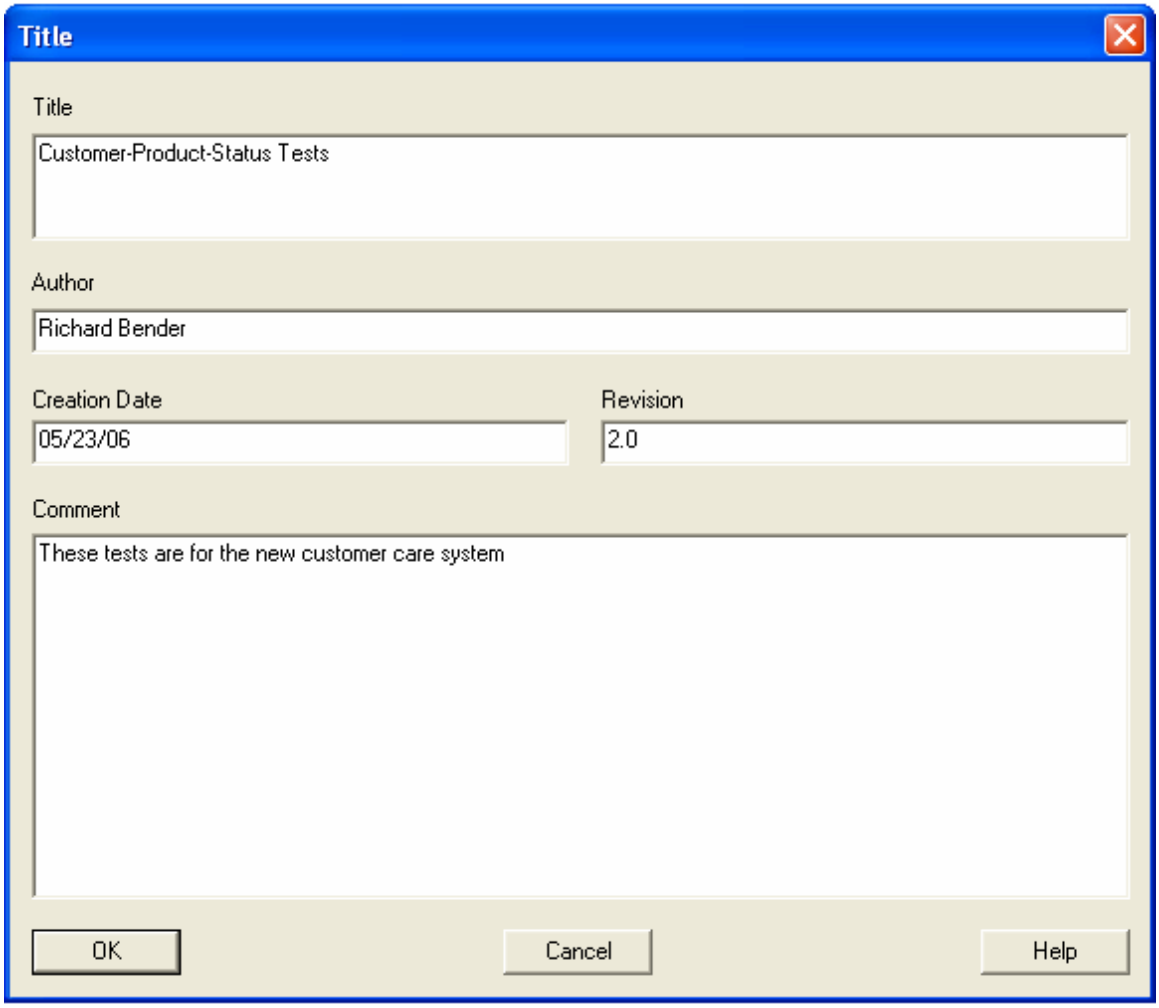


Quick Design Main Screen

Across the top you will see a number of options highlighted via green buttons: T, V, S, and C. These stand for Title, Variable, State, and Constraint. These are your building blocks. They are also available via a pull down menu under Inputs. In the next sections we will address how to complete each of these inputs.

3. Defining a Title

Selecting the Title option (T) will bring up a property sheet where you can define information to identify this group of tests you are designing.



Defining A Title

Whatever you enter in the Title section of this property sheet will be displayed as the header line on all of the reports generated by Quick Design. One suggestion is to use the name of the function you are testing, including the document, section number and version number / date of the source document. For example, instead of just having "Customer Profile Tests" you could have "Customer Profile Tests derived from Customer Care Functional Specification, version 2.1, section 5.1". There is no limit on the length for the title (actually there is a technical limit of 4 giga-bytes but you'll probably be able to define your titles in a bit less space).

The Author, Creation Date, and Revision sections are self explanatory. The Comment section is a free form text area for additional documentation about this set of tests. You might use this section to document a change history (in the off chance your specifications change from time to time during the development process).
[Note: See QD1.bqd in the Examples directory for this example.]

4. Defining Variables

Each variable represents an object to include in the tests. They might be inputs from the user interface (e.g. User-ID). They might be conditions of the system (e.g. Network-Status). They could be data from other applications being passed to this function. To define variables to Quick Design select the V button. This will cause the Variable property sheet to appear. You can also do this by placing the cursor in the left hand column and doing a right-click with the mouse.

Defining Variables

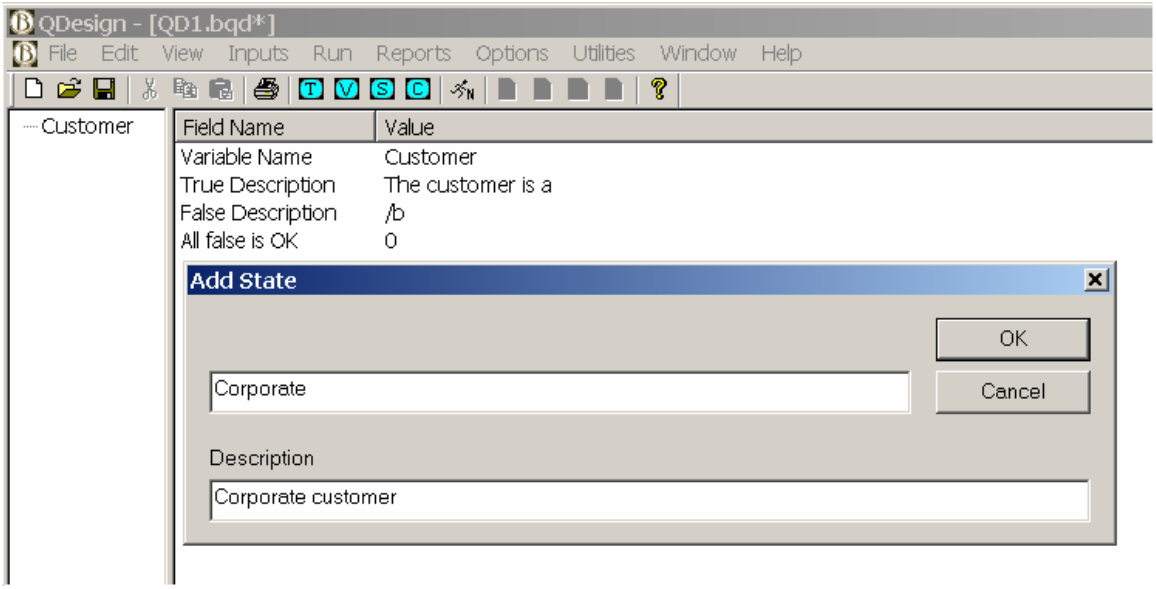
The Variable Name can contain any character. There is no limit to the length of the Variable Name.

For each Variable you can define a True Description and a False Description. The False Description only applies if it is OK for all of the States of that Variable to be false. Whatever you enter here will be concatenated with the State Description to form the description that will be displayed in your test scripts. In the above example the True Description is "The customer is a". This phrase will be completed via defining the State descriptions.

If the Variable must be at one of the States you will define for it then do not check the "it is OK for all states to be false box". If it is possible for none of the States to be true then check it. This will un-gray the False Description. Note that if the all-false state is possible for a Variable then you need a full False Description since there will be no corresponding State Description to concatenate it with. In the above example, if all-false is possible then a description "This is a non-customer" would be entered. There is no limit to the length of the Variable descriptions.

5. Defining States

To define States for a Variable, highlight the Variable and select the S button. This will cause the State Description property sheet to appear. You can also do this by highlighting a Variable and doing a right-click with the mouse.



Defining States

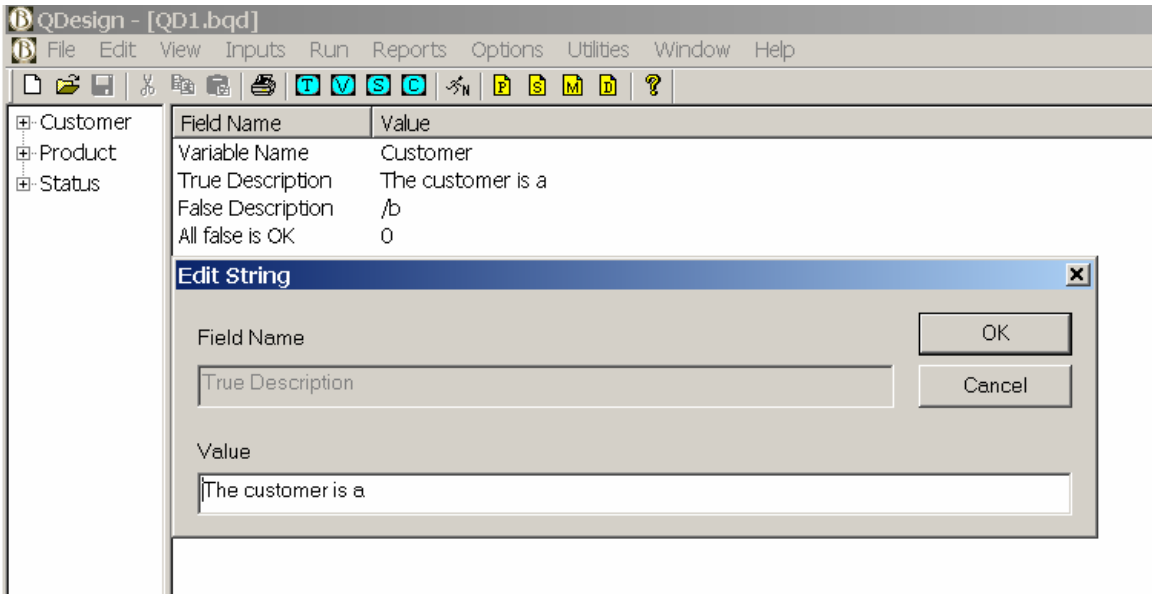
In this example we are adding a State to the Customer Variable called "Corporate". Its description is "Corporate customer". This will be appended to the Variable Description in the generated test scripts to create the phrase "The customer is a Corporate customer". How readable your test scripts are is dependent on how you craft these descriptions. Of course a big advantage of RBT in general is that it has a data base of these descriptions so you enter them only once and all your generated test scripts read absolutely consistent.

There is no limit to the length of the State Name or State Description.

You do want to take care in naming the Variables and States since these will show up in some of the reports – e.g. the Pairs Report. These reports will make more sense if the names are clear.

Once you have entered the Variable and State information, you might want to change it. To edit the data, select the Variable or State whose description you want to change. Its properties will appear in the information window. You then double click on the item you

want to change under the Field Name column. The will bring up an edit window. Make your changes and hit OK.



Editing Variable/State Descriptions

In this example the True Description is being edited. If you select the "All false is OK" option to edit, the only valid values are 1 (you can have the all-false state) or 0 (you cannot have the all-false state).

Selecting intelligent States to tests can be done in a number of ways. If you are familiar with the rules for generating functional variations from Cause-Effect Graphs you could use the variations deduced by looking at the operator in the specification. This does not require graphing.

Another alternative is to use equivalence class testing with boundary analysis (a technique originally developed by William Elmendorf in the mid-1960's at IBM's Poughkeepsie Labs). When you look at the domain (i.e. the set of all possible States) of a Variable, the domain is generally defined via a range, a list, or a category. [There are some other cases. For example when parsing strings you might say "starts with…". In effect this is just a variant of a list or category situation depending on how many things you are looking for.]

If the domain is defined via a range of values then select at least the following States to test: one in the middle, one at the highest of the valid values, one at the lowest of the valid values, one higher than the top valid value, and one lower than the lowest valid value. For example, if the valid range of values is from 10 to 100, you would choose: 50, 100, 10, 101, and 9.

If the domain is defined via a list then there are two situations. If the list is short, cover it completely. If the list is long, select a reasonable sample of values. For example, if the list had to do with highest level of education achieved and consisted of just: did not complete high school, completed high school, completed associate degree, completed bachelors degree, and completed advanced degree, you would make each value a State. If the list was of all the countries in the world, then just select a reasonable subset.

If the domain is defined via a single characteristic, then it is a two State problem. For example, if the specification says "for all female employees…" then define the Variable as Female and define the false state as male. Alternatively, define the Variable as Sex and define two states for it: Female and Male.

6. Defining Constraints

In the real world certain combinations of data just cannot happen. If they are not factored into the test design process, then you get test scripts you cannot physically create. If you toss out the whole test you usually will be tossing out valid portions/combinations that appear only in that test. The infeasible combinations occur for one of two reasons. One is the physical structure of the data. If the input is a one character field, then it can not contain two separate values at the same time. Another is editing that has occurred prior to getting to this function. If some combinations have been eliminated in this way you do not want them in the tests for this function. The invalid cases should have been tested in the function(s) that did the editing. The net is that you use constraints only when it is physically impossible for the data to violate them at this point in the system.

There are a number of constraint types you need to understand: **Exclusive**, **Inclusive**, **One**, **Requires**, and **Masks**. These constraints can be applied at the Variable level or at the State level.

Exclusive - If we say that two or more Variables/States are **Exclusive** then you can have at most one of them true in a test. Therefore, it is OK to have none of them true or to have one of them true. It is not possible to have two or more of them true.

Inclusive - If we say that two or more Variables/States are **Inclusive** then each test must have at least one of them true. It is OK to have two or more true in the same test. It is not possible to have all of them false in the same test.

One - If we say that two or more Variables/States are in a **One** constraint then each test must have one and only one of them true. You cannot have all of them false or two or more of them true in the same test.

This table summarizes the first three constraints:

	0	1	2 or more
Exclusive	OK	OK	Infeasible
Inclusive	Infeasible	OK	OK
One	Infeasible	OK	Infeasible

Requires - The first three constraints are bi-directional. That is if A is mutually exclusive with B then B is mutually exclusive with A. It works both ways. The **Requires** constraint is uni-directional. It defines a dependent relationship. For example, being pregnant true requires being female true. It does not mean, however, the reverse – being female true does not require being pregnant true. It is a one way constraint. The dependent Variable/State is called the constraint Subject. The Variable/State it depends on is called the constraint Object.

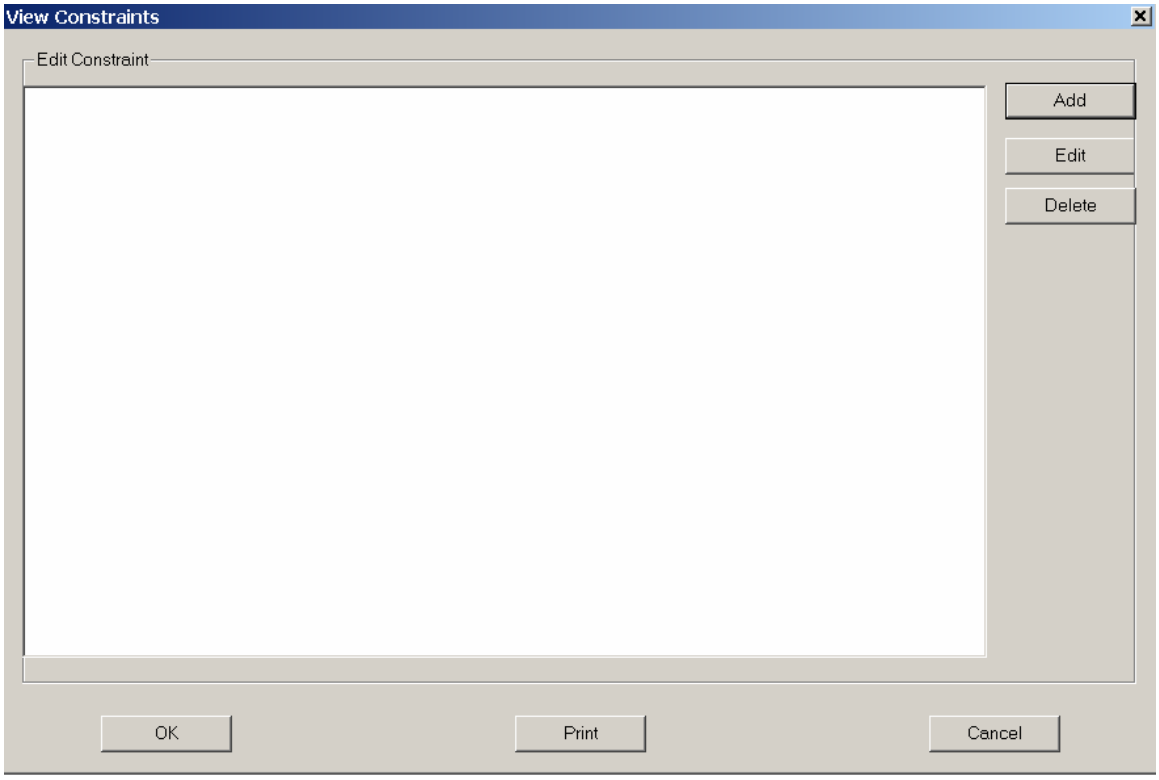
Masks - The last constraint is **Masks**. If we say that the state of a Variable/State **Masks** one or more other Variables/States we are saying that it causes these Variables/States to be ignored. This occurs for two reasons – attribute masks and sequencing masks.

If a Variable/State is an attribute of another Variable/State, then if that variable is false then its attributes do not exist. For example, let us talk about a person. We could say how tall the person is, what color eyes they have, how much they weigh, etc. For each variable – height, eye color, and weight – we could define multiple states. However, if it is possible for there not to be a person at all then we do not want to assign height, eye color, or weight attributes to this non-existent person. We would say that NOT Person MASKS height, eye color, and weight. Therefore, in any tests where the state of the Variable "Person" is false its attributes are neither true nor false – they are Masked. They do not exist in this situation. They will not be included in that test.

The other cause of **Masking** is the sequencing of processing. For example, let us say that at the user interface there are five Variables. We would then define each of them and their states. The editing of these variables might be such that as soon as the system finds an invalid one it stops processing and displays an error message. In other words, if the first Variable is invalid (i.e. false) then the system will ignore the other four Variables. We do not want to count as tested any of the States for these four Variables. The logic of the system branches around looking at the rest of them. In such a case we would say that the error State for the first Variable **Masks** all of the States for the other four Variables.

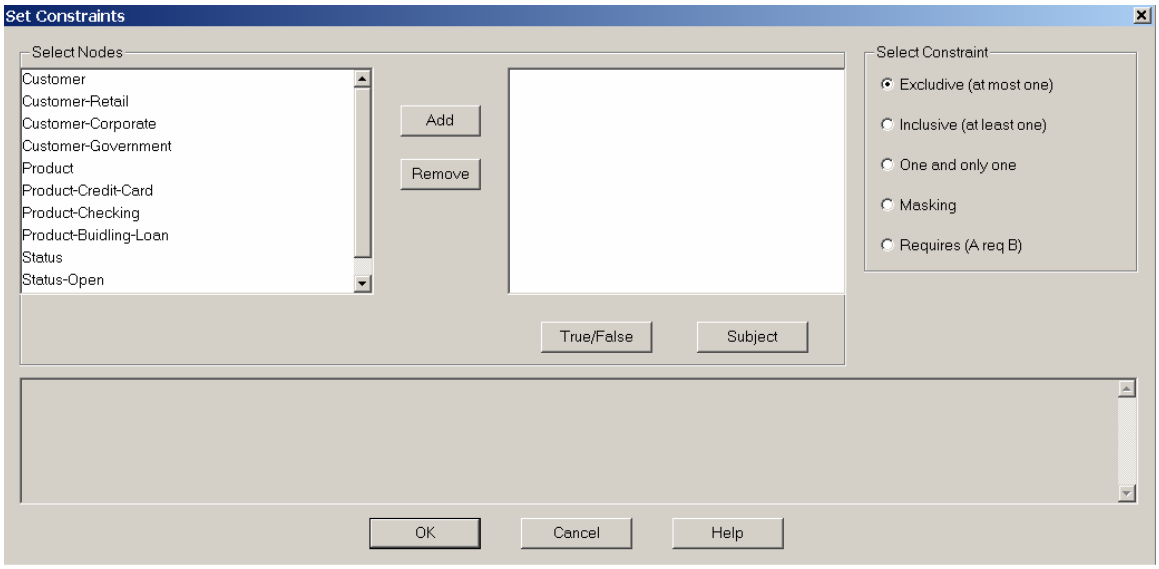
In the Mask constraint the Variable/State causing the other Variable/States to be ignored is the subject of the Mask. The Variable/State(s) being ignored are called the Object(s) of the Mask.

To define constraints select the C button from the menu. Alternatively, place the cursor in the left hand column, right click, and select Constraints. Either way the Constraint Dialog will appear.



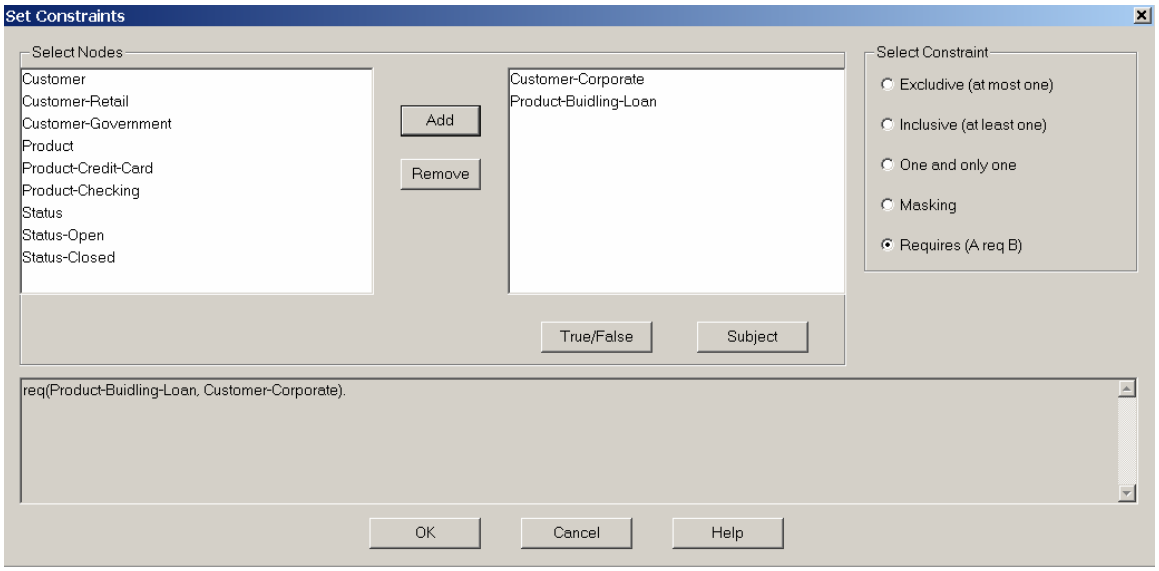
Constraint Dialog

You then select the Add button. This will populate the menu with the list of Variables/States and the list of Constraints.



Adding Constraints

You then select the Nodes – i.e. Variables or Variable/States to be included in the Constraint. You can select and add them in one by one. You can also add them in as a group. Just hold down the Ctrl button to select the set when you do this. You then select the Constraint type and hit add. This will generate the Constraint. In this example we have stated that to qualify for a Building Loan you must be a Corporate Customer – i.e. the Requires constraint. This implies that prior functions have edited out the other customer types when they tried to get a building loan.



Adding Constraints

For any of the Constraints the Variable or Variable/State in the constraint can be true or false. For example, you could specify that A, B and NOT C are mutually Exclusive. To

flip the true/false state of the Variable/State just highlight it in the middle window and hit the True/False button.

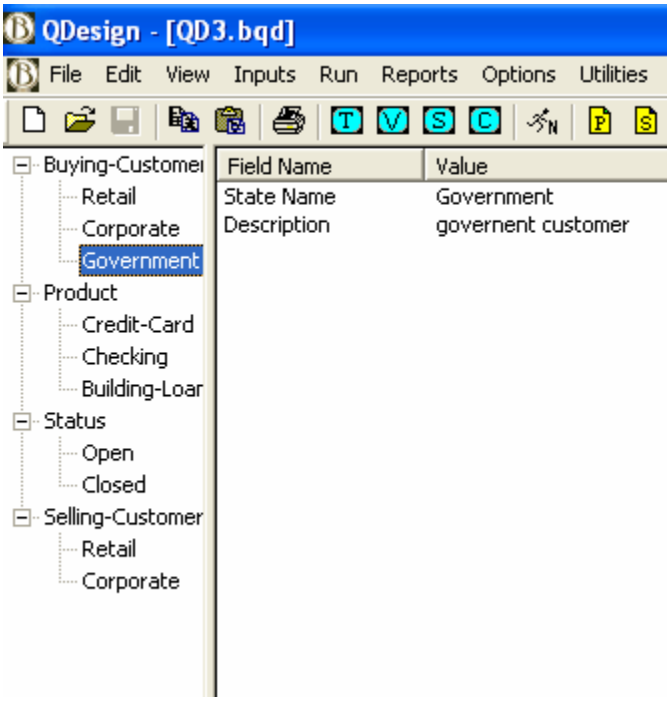
For the bi-directional Constraints the order in which they appear in the generated Constraint statement is not important. For the Requires and Masks constraints order is very important. If we say that A true Requires B true then A is the subject of the Requires constraint. If we say that NOT A Masks B, C, and D, then NOT A is the subject of the constraint. It must appear first in the Constraint statement.

Once you have created a constraint you can edit it. Again select the C option. When the initial Constraint dialog appears the list of already defined Constraints will appear in the window. Highlight the one you want to edit and select Edit. The second Constraint dialog will appear with the current values filled in. Make your changes and select OK.

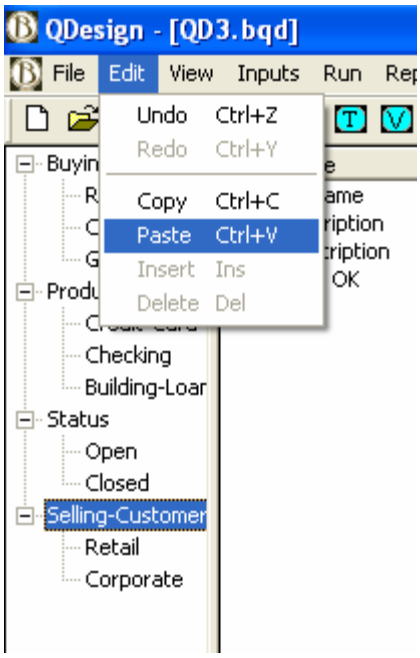
7. Editing The Quick Design Input

(See QD3.bqd in the Examples Directory for this section.)
It is not unusual to have multiple Variables with similar or identical States. You can easily copy states from one Variable to another. Let's say we have a Variable called Buying-Customer. It has three states: Retail, Corporate, and Government. We now create a new variable called Selling-Customer. We want the Selling-Customer to have the same States as the Buying-Customer. You can copy them in the following way:

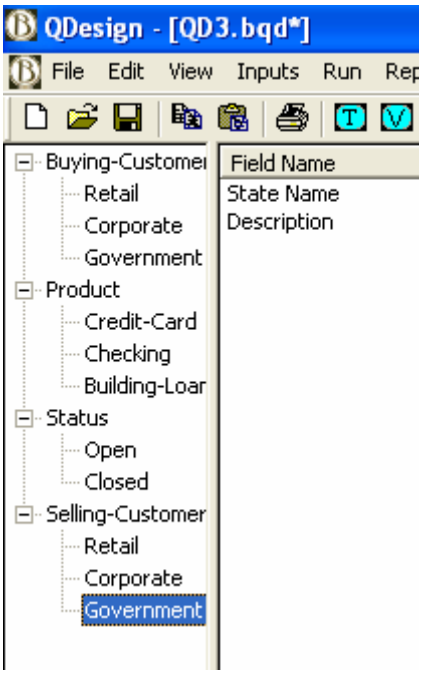
1. highlight the State to copy
2. select Edit/Copy
3. highlight the Variable into which you want to copy the State
4. select Edit/Paste.



Select the State to copy, Select Edit/Copy



Select the target Variable and select Edit/Paste

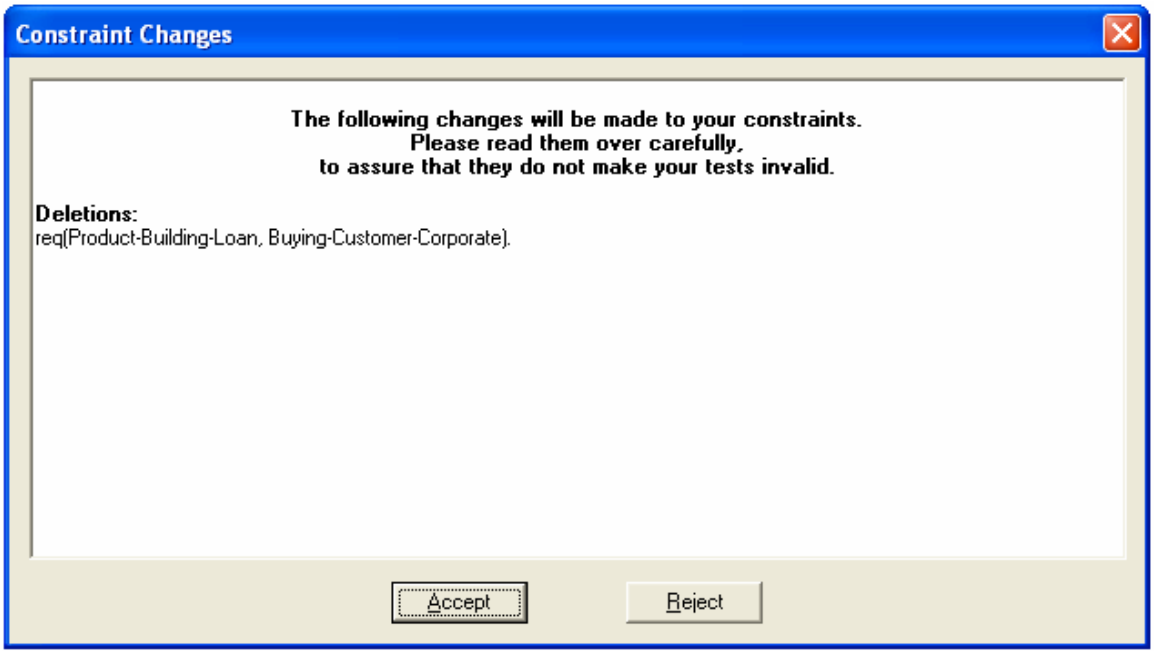


The State is now copied to the new Variable

When you delete Variables or States, Quick Design will check to see if any Constraints might be impacted. If you delete the Subject of a Requires or Mask Constraint it will prompt you that the Constraint will also be deleted. If the only remaining Object of a

Requires or Mask Constraint is being deleted it will prompt you that the Constraint will also be deleted. If only some of the Objects of a Constraint are being deleted the Constraint will be adjusted to reflect this.

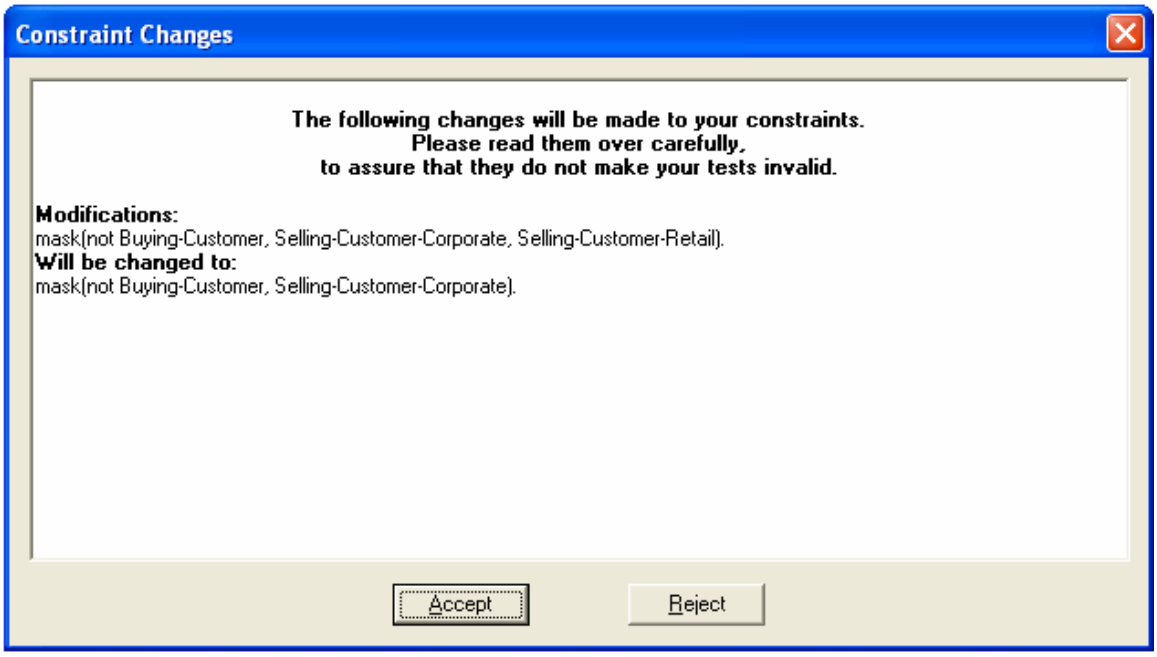
Let's say we have a Constraint: Product-Build-Loan Requires Buying-Customer-Corporate. We then want to delete the Building-Loan product. The following dialog will appear



Automatically deleting a Constraint

If you select "Accept" both the State and the Constraint will be deleted. If you select "Reject" no changes are made.

Let's say we have a Mask where if there is no Buying-Customer we cannot have the Selling Customer be a Corporate Customer or a Retail Customer (the Government Sellers will sell anyway – I guess to the tax payers). We now choose to delete just the Retail Selling Customer. The following dialog will appear:



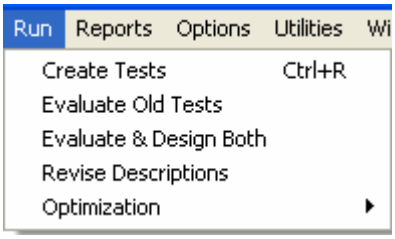
Automatically adjusting a Constraint

Selecting "Accept" will cause the Retail Selling Customer to be deleted and the Mask adjusted to eliminate reference to it. Selecting "Reject" will result in no changes be made.

This feature of automatically cross checking and adjusting Constraints is critical for keeping the various parts of a complex function in sync for testing. Even more importantly, this will help you will find cases where the specifications have been modified in one place but not updated elsewhere resulting in inconsistent specifications. Whenever Quick Design automatically informs you of the impact of a change you want to cross check that change with the various parts of the specifications that should also have been updated.

8. Designing and Maintaining Tests

Once you have defined your Variables, States, and Constraints, you are ready to have Quick Design design your tests. There are a number of test design options:



Test Run Options

Create Tests will design a new set of tests.

Evaluate Old Tests will take a set of tests that were previously designed and evaluate their coverage based on the current set of rules in the model.

Evaluate & Design Both start with an existing set of tests and determine their coverage. It will then add additional tests, if necessary, to complete the coverage.

Revise Descriptions allows you to change the names of Variables and States or change their descriptions without going through the full test case design process. It will update all of the reports to reflect the changes.

8.1 Create Tests

Quick Design generates the pairs to test and merges them into test cases. As the pairs are merged into tests, the Constraints you defined are applied to ensure that no test violates any of the Constraints.

The process of generating the pairs to test is straight forward. Each State of each Variable is paired with each State of the other Variables. In our example, the Customer Variable has three States; the Product Variable has three States, and the Status Variable has two States. Customer X Product generates 9 pairs. Customer X Status generates 6 pairs. Products X Status generates 6 pairs. This gives a total number of pairs of 21 pairs

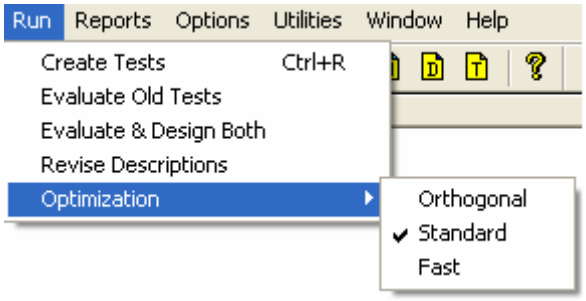
The pairs are initially generated by ignoring the Constraints. Then the Constraints are applied and the infeasible pairs are eliminated from the test design process. This occurs when the contents of a pair directly violate a Constraint or when there is no way to include a pair into any test combination without violating one or more Constraints.

You have three choices in designing tests: **orthogonal, standard**, and **fast**. In orthogonal pairs testing each pair exists in the set of tests the same number of times to the extent

possible. If pair one appears in four tests, all the other pairs will appear in four tests. This is affected by constraints. It is also affected by the number of states being different across variables. However, the goal is to produce as balanced a set of tests as possible.

In the standard and fast mode the goal is to include each feasible pair in at least one test while minimizing the total number of tests. For complex problems this leads to significant reductions in the number of tests. The difference between Standard and Fast is that for complex problems Standard will run slower but generally produce fewer tests. Fast will run faster but generally produce a few more tests than Standard. We suggest using fast for debugging your scripts and re-run using Standard to minimize the number of tests you need to build.
[Note: See QD2.bqd in the Examples directory. Run in Standard or Fast mode and in Orthogonal mode to see the difference in the number of tests versus the number of times a pair is included in a test.]

To select which mode of test design you want, just select the Run option and select the optimize option.



Optimization Options

To actually generate the pairs and merge them into tests select the "Create Tests" option in the Run pull down. If you want to design a new set of tests, the Create Tests option, you can also do this by selecting the run button.

Run Button

Quick Design will then design the tests as per your selected Optimization mode.

Designing tests can be an iterative process as you refine the Quick Design input until you are satisfied that the tests match the requirements. At that point you will proceed to implement the executable tests. This can be a non-trivial effort. It is important that the investment you make in building these tests is protected. If the rules change you do not want to go back and design a completely new set of tests. You would like to reuse as many of the old tests as possible. QD supports you in this effort.

Once you are happy with the set of tests you need to tell QD to remember these tests. This is done via **Utilities → Preserve Tests → Save Tests As**. This creates a separate test definition file that is not changed when you modify the QD input and rerun the test generator.

8.2 Evaluate Old Tests

Using QD4.bqd, we have the problem with Customers, Products, and Status that we had in QD1.bqd earlier. However, this version has removed the Constraint that Building Loans require that the customer is a Corporate Customer. The model has been run and the tests saved as QD4.bqt. This gives us the following tests:

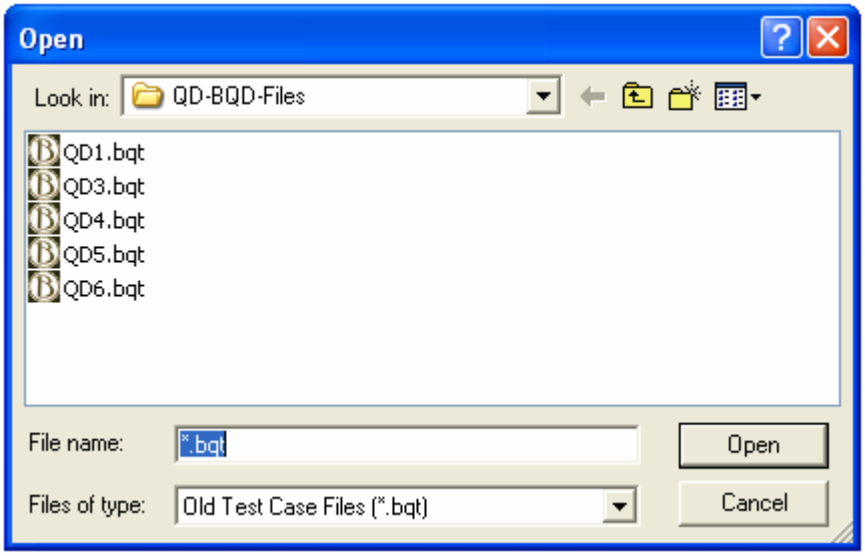
	TEST #1	TEST #2	TEST #3	TEST #4	TEST #5	TEST #6	TEST #7	TEST #8
Customer:Retail	F	F	F	F	T	F	F	T
Customer:Corporate	T	T	T	F	F	T	F	F
Customer:Government	F	F	F	T	F	F	T	F
Product:Credit-Card	F	F	T	F	T	F	T	F
Product:Checking	F	F	F	T	F	T	F	T
Product:Building-Loan	T	T	F	F	F	F	F	F
Status:Open	T	F	T	F	F	F	T	T
Status:Closed	F	T	F	T	T	T	F	F

Test Definition Matrix

We then add back in the constraint that Building Loans require that it is a corporate customer. We want to see what impact this has on the set of tests that were created before this constraint was present. To do this we choose the

Run → Evaluate Old

option to see what this does to our tests. This will prompt you to select which set of old tests to evaluate.



Select the QD4.bqt tests. [Note: another way to select which old tests you want to evaluate is via **Utilities → Preserve Tests → Open Old Tests**. This will make that set the active set.] When you bring up the Test Definition Matrix after the run you get the following:

Click here to Evaluate Old Tests	TEST#1-OLD	TEST#2-OLD	TEST#3-OLD	TEST#4-OLD	TEST#5-OLD	TEST#6-OLD	TEST#7-OLD	TEST#8-OLD	TEST#9-OLD
Customer:Retail	T	F	F	F	T	F	T	F	F
Customer:Corporate	F	T	F	T	F	F	F	T	F
Customer:Government	F	F	T	F	F	T	F	F	T
Product:Credit-Card	F	T	F	F	T	T	F	F	F
Product:Checking	F	F	T	T	F	F	T	F	F
Product:Building-Loan	T	F	F	F	F	F	F	T	T
Status:Open	T	T	F	T	F	T	T	F	F
Status:Closed	F	F	T	F	T	F	F	T	T

Evaluate Old Tests

The shaded columns represent tests that are now invalid since they violate a constraint. The same information will show up in the Script report.

If a Variable or a Variable-State that was in the old tests had been deleted from the model, this would also be highlighted in the Test Definition Matrix.

	TEST#1-OLD	TEST#2-OLD	TEST#3-OLD	TEST#4-OLD	TEST#5-OLD	TEST#6-OLD	TEST#7-OLD	TEST#8-OLD
Customer:Retail	F	F	F	F	T	F	F	T
Customer:Corporate	T	T	T	F	F	T	F	F
Customer:Government	F	F	F	T	F	F	T	F
Product:Credit-Card	F	F	T	F	T	F	T	F
Product:Checking	F	F	F	T	F	T	F	T
Product:Building-Loan	T	T	F	F	F	F	F	F
Status:Open (Deleted)	-	-	-	-	-	-	-	-
Status:Closed (Deleted)	-	-	-	-	-	-	-	-

Variable-States Deleted From Model

In this case we have deleted the Variable "Status" and its States. The Matrix tells us that the old tests need to be modified to reflect this. Similarly if new Variables or Variable-States have been added to the model, Quick Design will supplement your tests as needed.

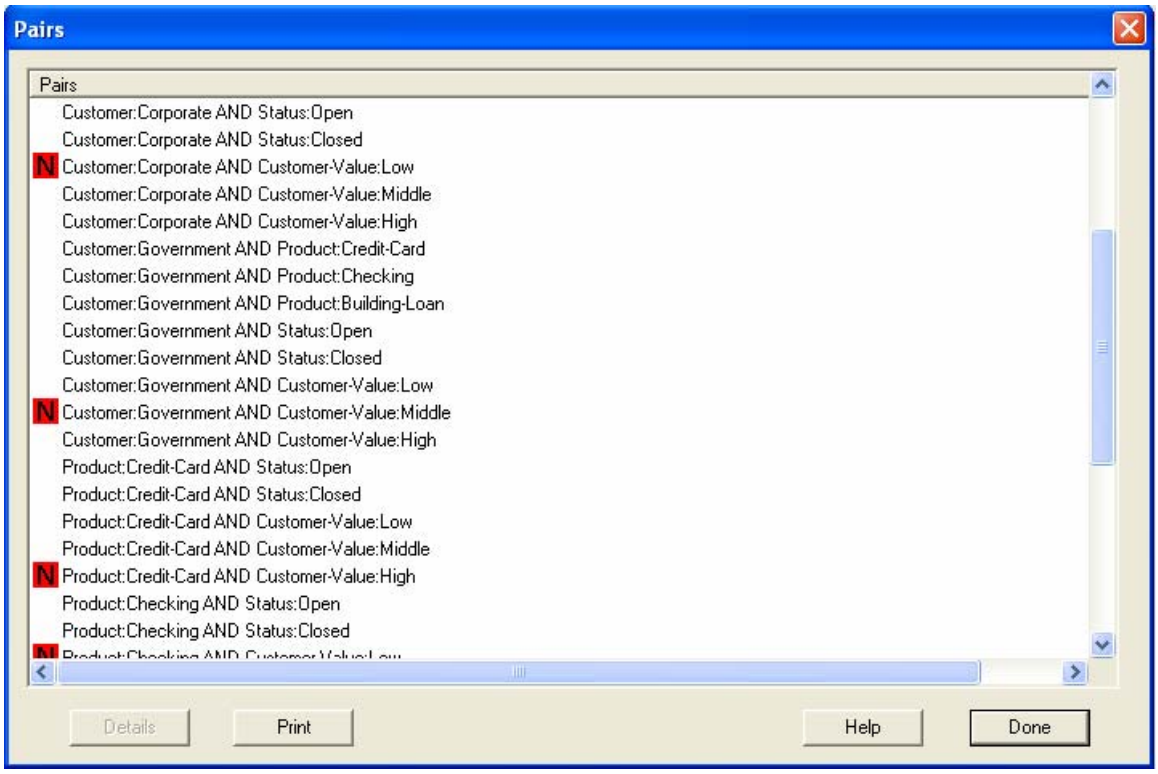
8.3 Evaluate & Design Both

Using QD4.bqd again, without the constraint to keep things simple, let us change the model to add a new variable called Customer Value. It has three states: low, middle, and high. (We will save this version as QD5.bqd.) None of our original tests had this variable. However, we can still use them. QD will tell us what needs to be done to supplement the old test cases. We run using the old test cases and get the following:

	TEST#1-OLD	TEST#2-OLD	TEST#3-OLD	TEST#4-OLD	TEST#5-OLD	TEST#6-OLD	TEST#7-OLD	TEST#8-OLD	TEST#9-OLD
Customer:Retail	T	F	F	F	T	F	T	F	F
Customer:Corporate	F	T	F	T	F	F	F	T	F
Customer:Government	F	F	T	F	F	T	F	F	T
Product:Credit-Card	F	T	F	F	T	T	F	F	F
Product:Checking	F	F	T	T	F	F	T	F	F
Product:Building-Loan	T	F	F	F	F	F	F	T	T
Status:Open	T	T	F	T	F	T	T	F	F
Status:Closed	F	F	T	F	T	F	F	T	T
Customer-Value:Low (New)	T	F	F	F	F	T	F	F	T
Customer-Value:Middle (New)	F	T	F	T	T	F	F	F	F
Customer-Value:High (New)	F	F	T	F	F	F	T	T	F

Old Tests Enhanced

Notice that the Customer Value variable and its states have been added to the existing tests. You just make the corresponding changes to your existing tests to bring them in sync with the new rules. However, we still have a problem. Not all of the Pairs are covered by these tests. The pairs with the red N (for not covered) are not in any of the existing tests.



Pairs Not Covered

The next step is to rerun using the **Run → Evaluate & Design Both** option. This will use the existing tests to cover as much as possible but will also add any additional tests needed to complete coverage. We now get the following tests:

	TEST#1-OLD	TEST#2-OLD	TEST#3-OLD	TEST#4-OLD	TEST#5-OLD	TEST#6-OLD	TEST#7-OLD	TEST#8-OLD	TEST#9-OLD	TEST#10-NEW	TEST#11-NEW	TEST#12-NEW
Customer:Retail	T	F	F	F	T	F	T	F	F	F	F	T
Customer:Corporate	F	T	F	T	F	F	F	T	F	T	F	F
Customer:Government	F	F	T	F	F	T	F	F	T	F	T	F
Product:Credit-Card	F	T	F	F	T	T	F	F	F	F	F	T
Product:Checking	F	F	T	T	F	F	T	F	F	T	F	F
Product:Building-Loan	T	F	F	F	F	F	F	T	T	F	T	F
Status:Open	T	T	F	T	F	T	T	F	F	F	F	T
Status:Closed	F	F	T	F	T	F	F	T	T	T	T	F
Customer-Value:Low (New)	T	F	F	F	F	T	F	F	T	T	F	F
Customer-Value:Middle (New)	F	T	F	T	T	F	F	F	F	F	T	F
Customer-Value:High (New)	F	F	T	F	F	F	T	T	F	F	F	T

Old Tests Enhanced and Supplemented

Notice that the additional tests are annotated as "NEW". You can then save off this set of tests and get ready for the next cycle.

8.4 Revise Descriptions

Once you have finished designing your tests you might decide that there was a better way to describe a Variable or State. You can change descriptions and update all of your documentation without having to go through the full test design process. To do this, select **Run → Revise Descriptions.** This is useful for very large models which can take a while to run. You may make the following changes:

Change a Variable's name
Change a State's name
Change a Variable's description
Change a State's description

However, any of the following changes will cause the Revise Descriptions to be grayed out:

Adding or deleting a Variable
Adding or deleting a State

Adding, deleting, or changing a Constraint

Changes to a Variable or State name will also be reflected in any Constraint which contains it.

8.5 Editing Old Tests

Once you save your tests you can edit them. You can change the test ID or even modify the state of the Variable-States. To edit your tests you select:

Utilities → Preserve Tests → Open Old Tests

This activates the test editor feature. You then open the test file you want to edit:

Utilities → Preserve Tests → Edit Old Tests

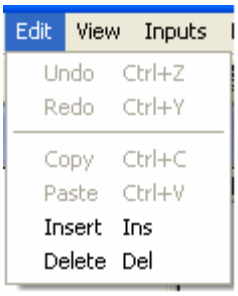
This brings up the test cases so you can edit them:

Old Tests Editor

Select the test and the Variable-State you want to edit. You can then change the entry. The only valid entries are T (true), F (false), and M (masked).

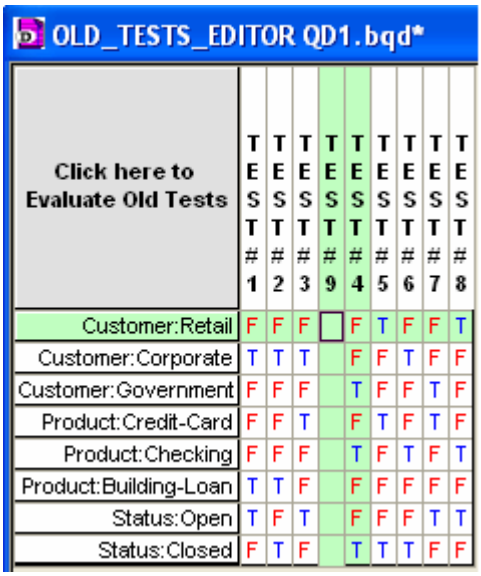
You can use this feature to tell Quick Design about tests you already have from a previous effort, even though they were not designed by QD. You go to:

Edit → Insert



Insert/Delete Tests

QD will insert a blank test case to the left of whatever test is currently selected.



Insert New Test

It will assign the test name with the next number as its ID – e.g. if you had 8 tests, the new test will be named Test #9. You then use the editor to tell QD the state of each Variable-State.

After you have defined your existing tests, then use the Run Old option to evaluate how much they cover. Check the Coverage Matrix Report. This will quickly show you if any are any Untested pairs. Also, you can see if you have any redundant tests. A pair covered by only one test is denoted with a "#". If a variation is in two or more tests it is marked with an "X". Any test designed by QD in Optimization mode has at least one pair that is not covered by another test. If you have tests that all "X"'s, then there is some redundancy.

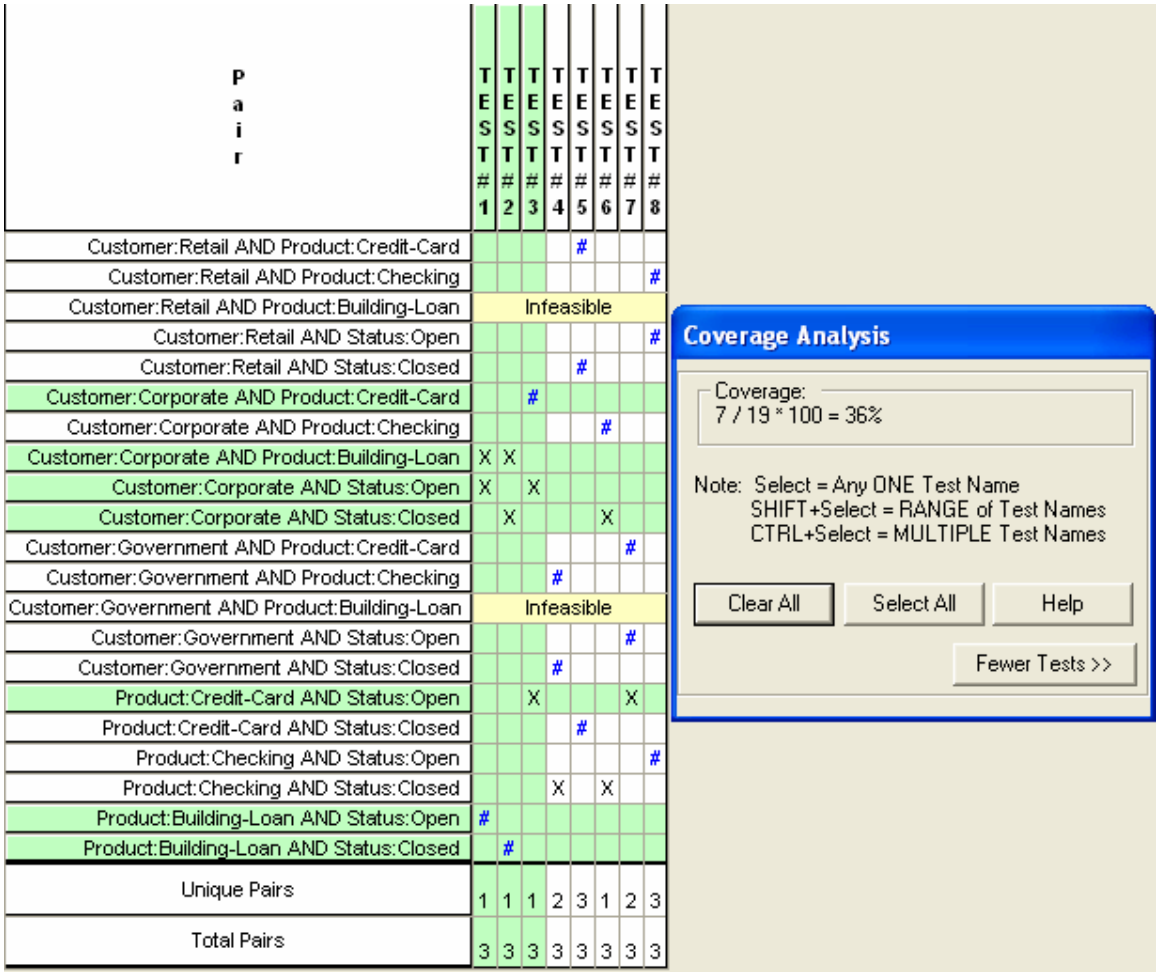
Using the information in the Coverage Matrix you can decide which tests to delete while still keeping the same level of test coverage. However, the "redundant" tests might be

there because of design dependent or code dependent considerations. Therefore, take care before deleting tests from your existing test library.

8.6 Coverage Analysis

The Coverage Analysis utilities can do two things. First, they can calculate what the coverage is for any subset of the tests. Second, they can determine what the optimal coverage is for any subset of the tests.

Selecting the Coverage Analysis Utility (**Utilities → Coverage Analysis**) will bring up the Coverage Matrix and a dialog showing the calculated coverage.



Pair	TEST #1	TEST #2	TEST #3	TEST #4	TEST #5	TEST #6	TEST #7	TEST #8
Customer:Retail AND Product:Credit-Card					#			
Customer:Retail AND Product:Checking								#
Customer:Retail AND Product:Building-Loan	Infeasible							
Customer:Retail AND Status:Open								#
Customer:Retail AND Status:Closed					#			
Customer:Corporate AND Product:Credit-Card			#					
Customer:Corporate AND Product:Checking						#		
Customer:Corporate AND Product:Building-Loan	X	X						
Customer:Corporate AND Status:Open	X		X					
Customer:Corporate AND Status:Closed		X				X		
Customer:Government AND Product:Credit-Card							#	
Customer:Government AND Product:Checking				#				
Customer:Government AND Product:Building-Loan	Infeasible							
Customer:Government AND Status:Open							#	
Customer:Government AND Status:Closed				#				
Product:Credit-Card AND Status:Open			X				X	
Product:Credit-Card AND Status:Closed					#			
Product:Checking AND Status:Open								#
Product:Checking AND Status:Closed				X		X		
Product:Building-Loan AND Status:Open	#							
Product:Building-Loan AND Status:Closed		#						
Unique Pairs	1	1	1	2	3	1	2	3
Total Pairs	3	3	3	3	3	3	3	3

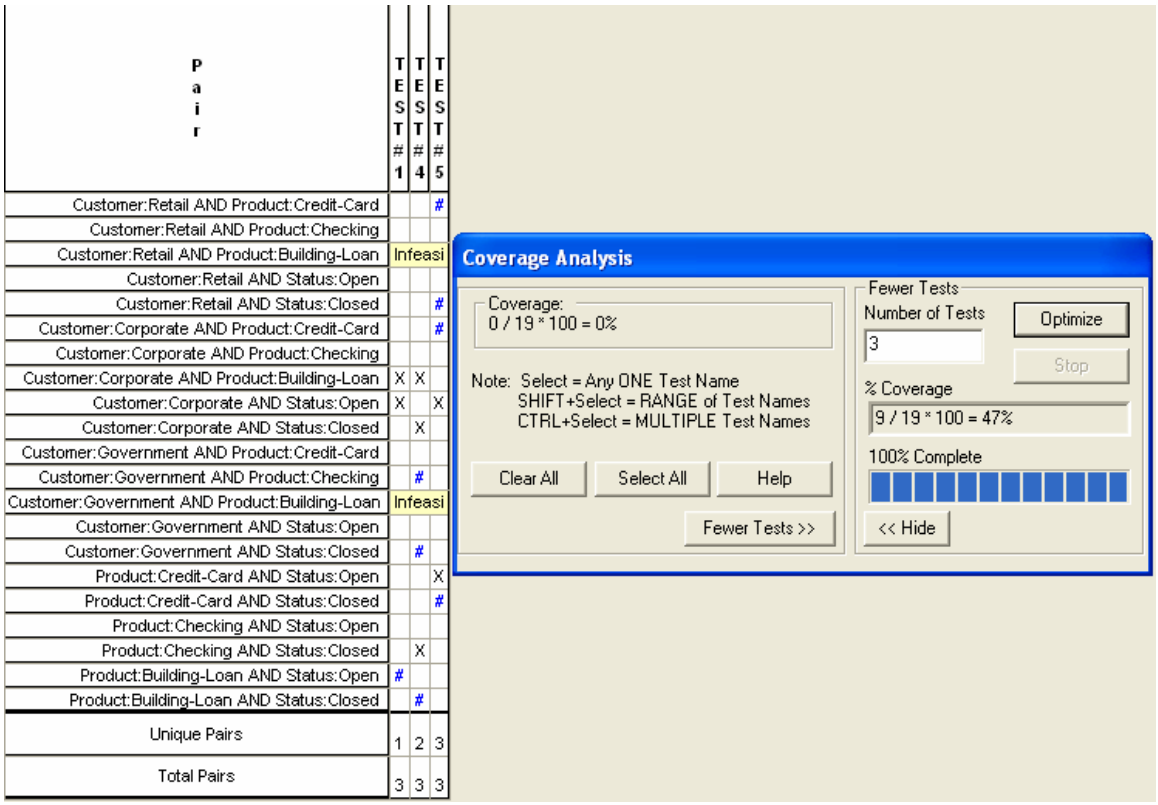
Coverage Analysis Dialog

In this example (using QD1) three tests have been marked as being successful – the ones in green. To mark a test has having been successful just click on that column. The dialog

follows MS Windows conventions. If you select a column it highlights. If you select another column, only the new column is highlighted. To select more than one test keep the CTRL button pressed as you select additional ones. To select a range of columns keep the SHIFT button pressed while you select the first and last column in the range.

With these three of the seven tests the current status is Coverage = 36%. This means that 36% of the feasible pairs are in test cases that were successful.

Notice on the dialog the **Fewer Test >>** option. Selecting this brings up a supplemental dialog:



Fewer Tests Dialog

This feature allows you to enter in a number less than or equal to the number of total tests and have QD determine which is the optimal subset of tests – i.e. which tests would give you the greatest possible coverage.

The overall coverage feature is primarily used to measure and report test status. As a tester I love being able to tell Management, quantitatively, the status of testing. We take these numbers for each function and put them on spreadsheets. We can then calculate the overall coverage for the system. If Management wants to deploy the application prematurely, we ask them to sign these spreadsheets so we have a record of the status at the time of deployment. If we said something was tested and defects are found in production in that area, then that is our problem. However, if we made it clear that

something was not yet finished testing and defects are later found, then that is Management's problem.

The Fewer Tests feature is used for two purposes. First, if Management is not giving you enough time to build and run all of the necessary tests, you can use this feature to select the best set of tests possible within the constraints you have been given. In the above example, if there is only time for creating three of the seven tests then we should choose tests 1, 4, and 5. We can also take this information to Management and explain ahead of time what the best we do will be. If 47% Coverage is the best we can do, maybe this is not a good decision to limit the time we need.

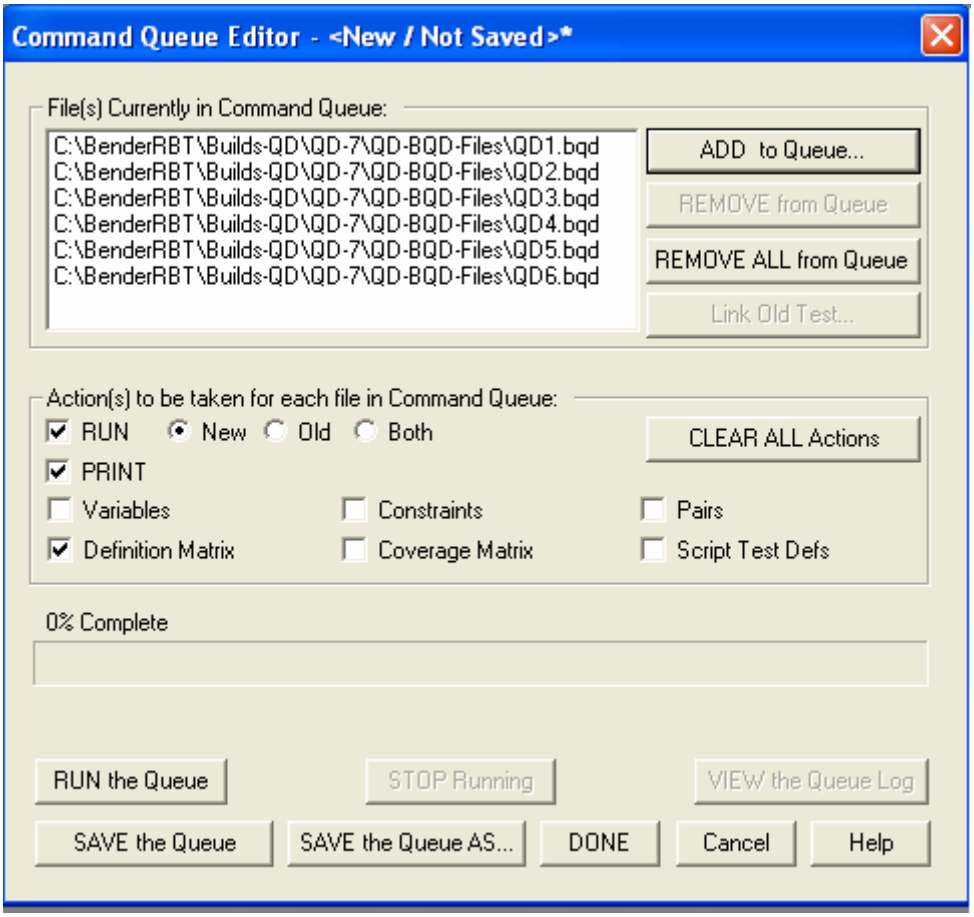
The second use is to optimize the testing effort even where we do have time to create and run all of the needed tests. You would use this feature to decide which tests to build and run first. That will give you the greatest coverage in the shortest amount of time.

You can save this set of tests using **Utilities → Preserve Tests → Save As**.

9. Creating/Running A Queue

Quick Design also allows you set up a queue of BQD files to run automatically one after the other. This is very convenient if you have a number of them that run for a while and you want to run all of them overnight.

To create a queue just select **File → Command Queue → New**. Add the files you wish to run as a batch and select the run and print options you want.

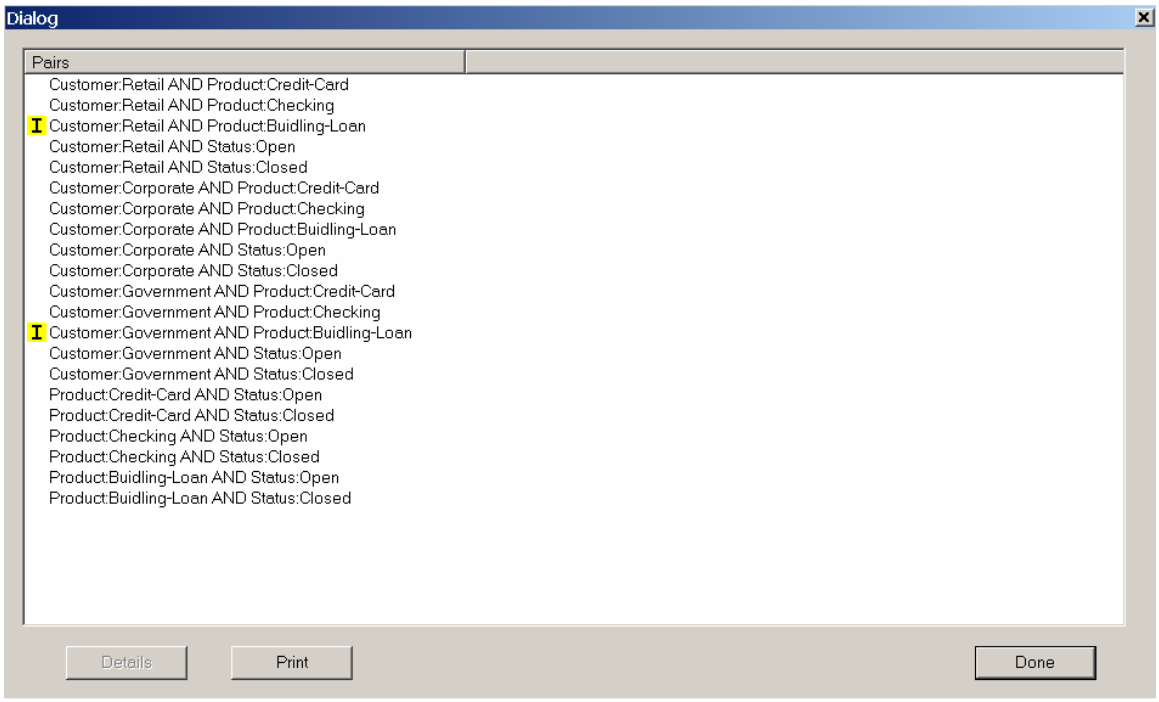


Queue Manager

10. Reports

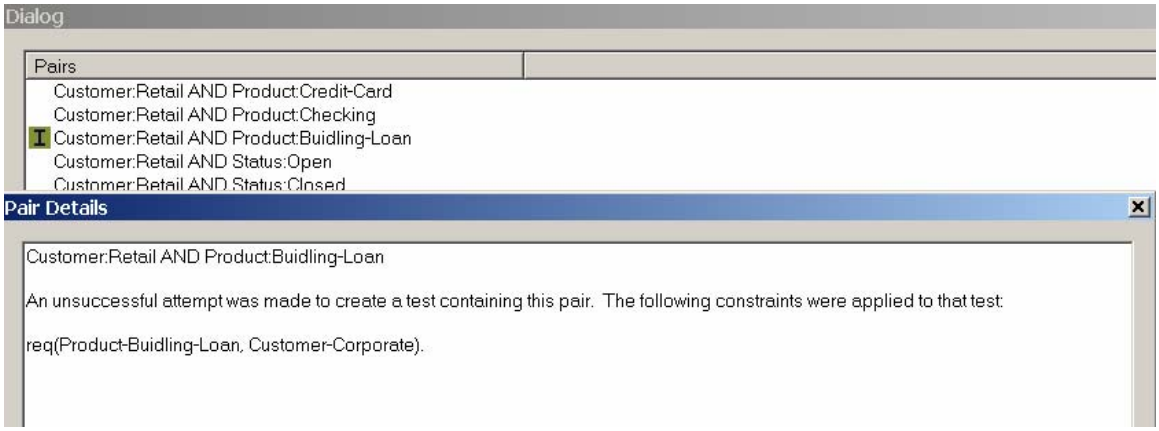
Once Quick Design is run it produces a number of reports: Pairs, Scripts, Coverage Matrix, Test Definition Matrix, and Statistics.

10.1 Pairs Report



Pairs Report

The **Pairs Report** lists all of the pairs and denotes which ones are infeasible due to the constraints. This is done via the "I" with a yellow background in the first position. If you want to understand why a given pair is infeasible, just highlight it and hit the Details button. Quick Design then displays the constraint(s) that caused it to be infeasible.

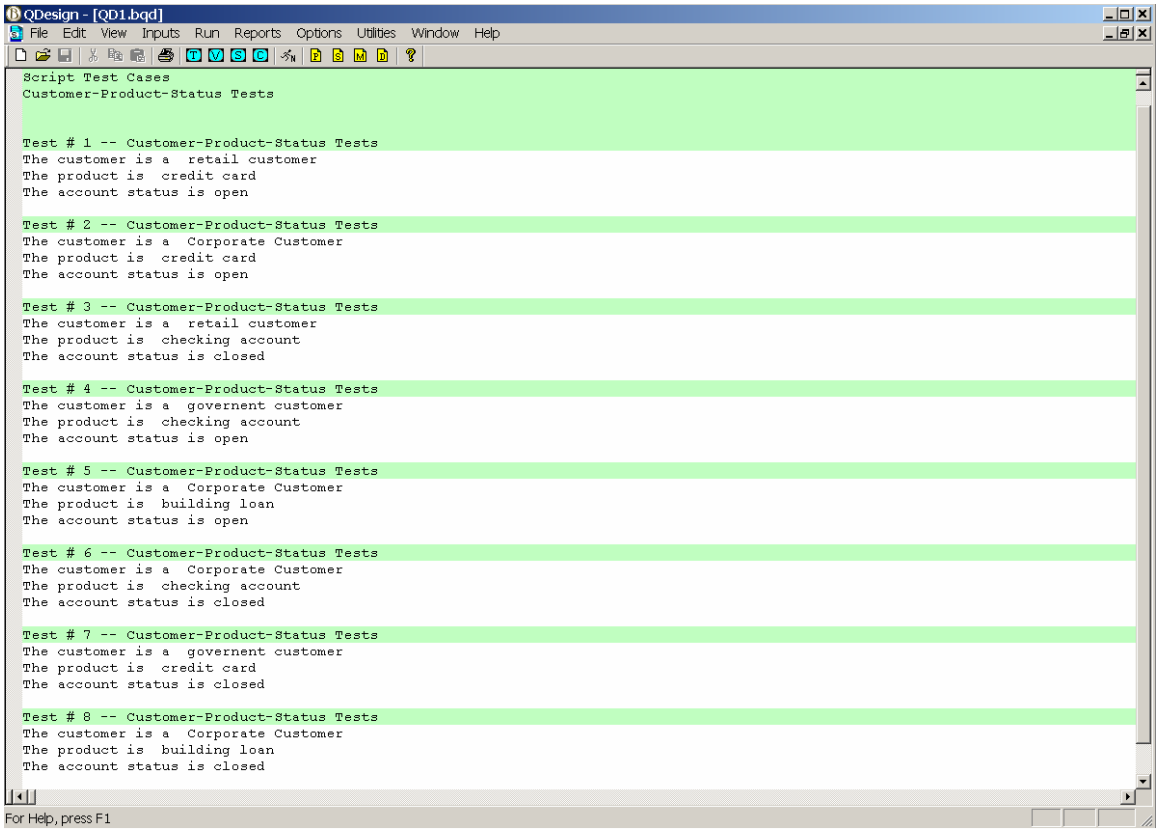


Infeasible Pairs Analysis

In this example the pair consisting of a retail customer having a building loan is infeasible because we restricted building loans to corporate customers.

10.2 Script Report

The **Script Report** is the text definition of the test cases. How readable these scripts are is totally dependent on how careful you are with the Variable/State definitions.



Script Report

10.3 Coverage Matrix

The **Coverage Matrix** identifies which pairs are in which test case. If a pair is in only one test then it is denoted with a #. If it is in two or more tests it is denoted with an X. When Quick Design is in Optimize mode every test will have at least one pair unique to it. If the tests were designed under the orthogonal rules then all of the entries might be X's.

The report also notes which pairs were infeasible.

QDesign - [QD1.bqd]

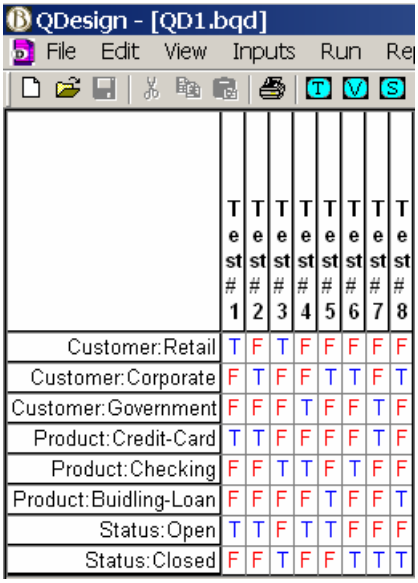
File Edit View Inputs Run Reports Options Utilities Window Help

Pair	Test #1	Test #2	Test #3	Test #4	Test #5	Test #6	Test #7	Test #8
Customer:Retail AND Product:Credit-Card	#							
Customer:Retail AND Product:Checking			#					
Customer:Retail AND Product:Buidling-Loan	Infeasible							
Customer:Retail AND Status:Open	#							
Customer:Retail AND Status:Closed			#					
Customer:Corporate AND Product:Credit-Card		#						
Customer:Corporate AND Product:Checking						#		
Customer:Corporate AND Product:Buidling-Loan					X			X
Customer:Corporate AND Status:Open		X			X			
Customer:Corporate AND Status:Closed						X		X
Customer:Government AND Product:Credit-Card							#	
Customer:Government AND Product:Checking				#				
Customer:Government AND Product:Buidling-Loan	Infeasible							
Customer:Government AND Status:Open				#				
Customer:Government AND Status:Closed							#	
Product:Credit-Card AND Status:Open	X	X						
Product:Credit-Card AND Status:Closed							#	
Product:Checking AND Status:Open				#				
Product:Checking AND Status:Closed			X			X		
Product:Buidling-Loan AND Status:Open					#			
Product:Buidling-Loan AND Status:Closed								#
Unique Pairs	2	1	2	3	1	1	3	1
Total Pairs	3	3	3	3	3	3	3	3

Coverage Matrix

10.4 Test Definition Matrix

The next report is the **Test Definition Matrix**. It presents a compact definition of the tests. For each test it specifies the state of each Variable/State. Each Variable-State will be either T (True), F (False), or M (Masked).



	Test #1	Test #2	Test #3	Test #4	Test #5	Test #6	Test #7	Test #8
Customer:Retail	T	F	T	F	F	F	F	F
Customer:Corporate	F	T	F	F	T	T	F	T
Customer:Government	F	F	F	T	F	F	T	F
Product:Credit-Card	T	T	F	F	F	F	T	F
Product:Checking	F	F	T	T	F	T	F	F
Product:Buidling-Loan	F	F	F	F	T	F	F	T
Status:Open	T	T	F	T	T	F	F	F
Status:Closed	F	F	T	F	F	T	T	T

Test Definition Matrix

Most of the reports can be exported. Just bring up the report, select File, and then select Export. The Script report is exported as a text file. It can then be pulled into any text processor – e.g. MS Word. We do not export the Pairs Report.

The Coverage matrix and the Test Definition matrix are exported as comma delimited files. These can then be read by Excel.

10.5 Statistics Report

The final report is the **Statistics Report**. It documents the number of pairs, number of tests, the theoretical maximum number of tests, and the run time.

```
Test Statistics
QD2 - Large Combinations
Input Graph Filename:   C:\BenderRBT\Documentation\Bender-RBT
File Last Saved:    11/20/2003  6:48:24 PM

Design Tests Last Run:  5/26/2006  5:35:11 PM
BenderRBT Release:       1.1(200)

Run:  Synthesis of New Tests
Run Optimization is set to:  Standard

Number of pairs:  702
Number of infeasible pairs:  0


Number of new test cases defined:  19

Number of tested pairs:       702
Number of feasible pairs:     702
Percentage of coverage of feasible pairs:
    702/702*100 = 100%

Number of tested pairs:       702
The THEORETICAL maximum number of test cases is:  1,594,323

The number of test cases generated by BenderRBT is:  19
The test case compression ratio is:
    1,594,323/19 = 83,912 : 1

The feasible pairs to test case compression ratio is:
    702/19 = 36.95 : 1

QuickDesign Elapsed Time =  00:00:01
```

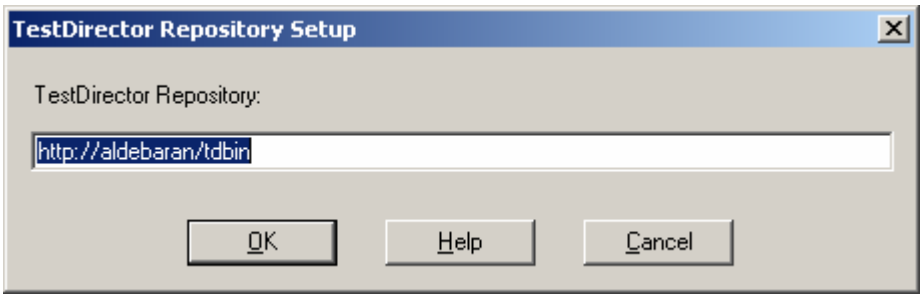
Statistics Report

11. Exports

The test cases designed by Quick Design can be exported to various third party test managers. Currently we support Mercury Interactive's TestDirector and Sirius-SQA's TestExplorer.

11.1 Export to Mercury Interactive's TestDirector

You first need to tell QD where to find the TestDirector repository. Select **Options → TestDirector Repository**. The following dialog will appear:

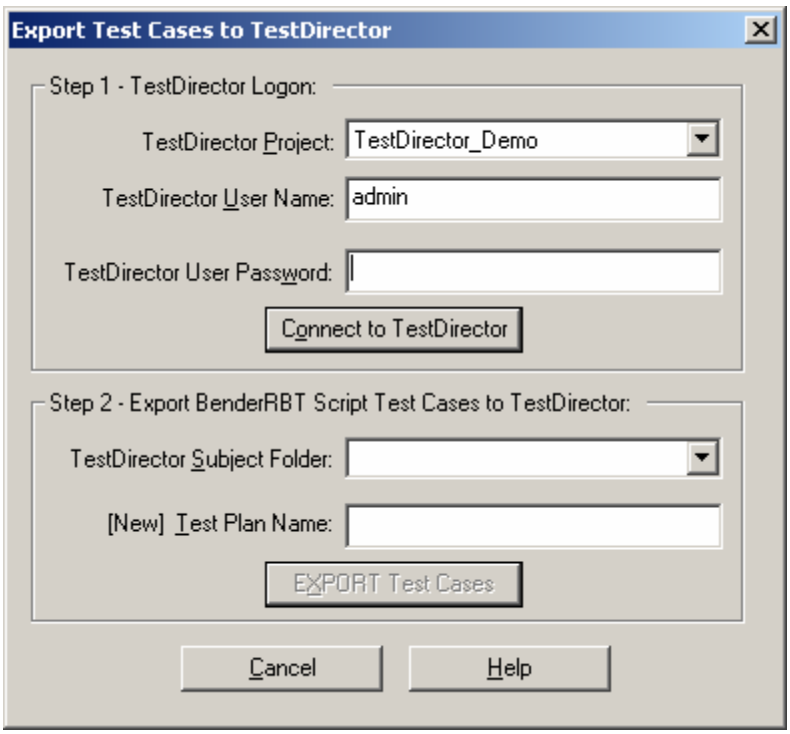


Path to TestDirector

Enter the fully qualified path name.

Then to export the test scripts to TestDirector select the **Utilities → Export to Test Director** option. Quick Design populates the test planning section in Test Director.

Warning: When you export your tests to TestDirector a folder will be created. Your TD privileges must be set up in such a way that you are allowed to create folders. Otherwise the export will fail. When it does fail for this reason TestDirector does not give back any specific return code that allows us to assist you in debugging the problem.



Export to TestDirector

11.2 Export to Sirius-SQA's TestExplorer

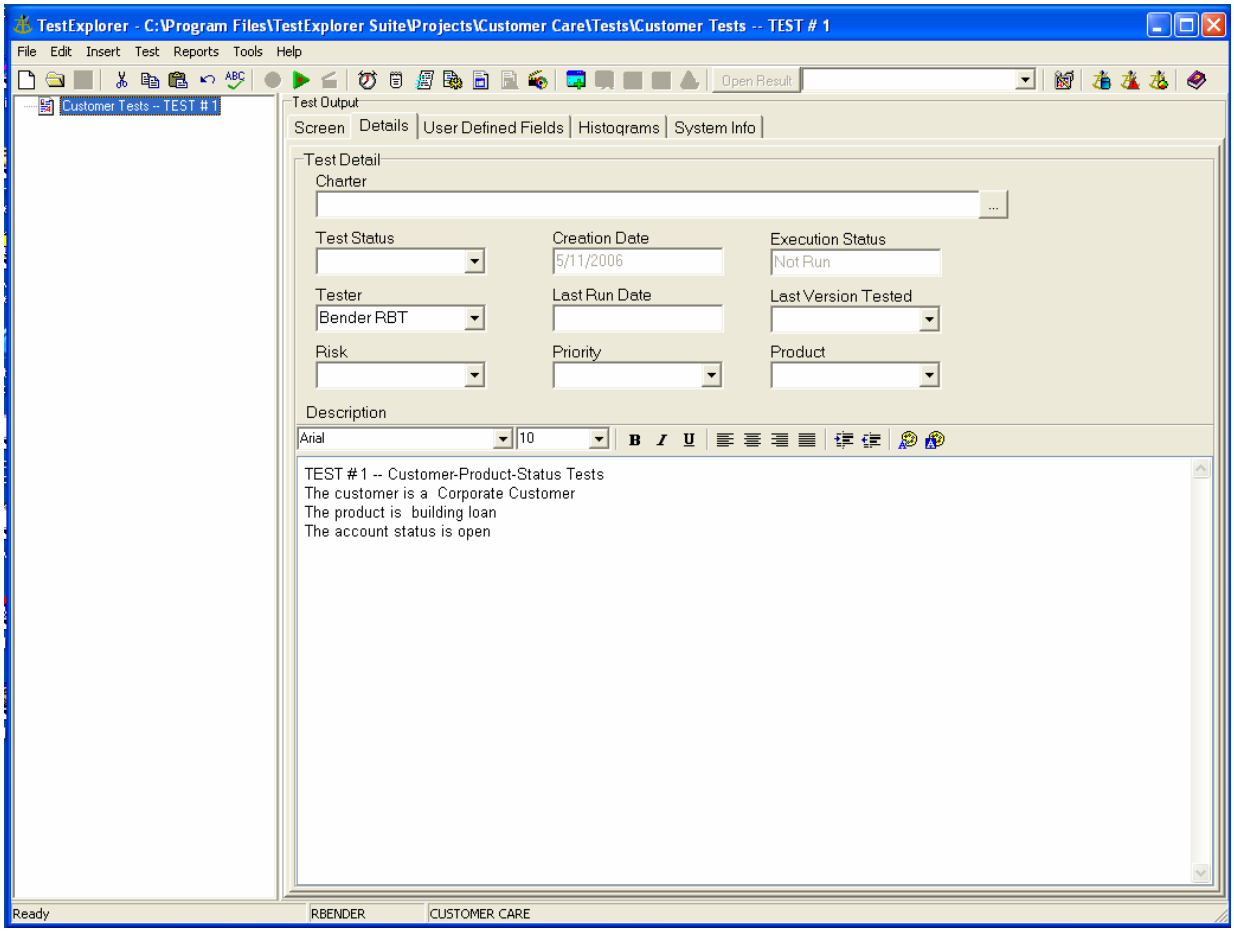
This feature works the same way as the export to TestDirector. TestExplorer is a test management tool aimed at manual testing. More information can be found about the product and the company at http://www.Sirius-SQA.com.

Unlike TestDirector, you do not need to tell RBT where TestExplorer is located. Each desktop installation of TestExplorer has a configuration file in the system directory that contains the path to the root drive where all projects reside. By default, the database is on the local drive, but it can reside on any network shared drive - in team environments it likely would. However, the configuration file lives on the local desktop, so each desktop knows where its database is. Consequently, different teams can have different databases as well. The dll is registered on the computer where RBT is located and finds this configuration file, which tells RBT where the actual database is. This configuration file is created when TestExplorer is installed. The implication is that for the integration to work, TestExplorer has to be installed and properly configured on the desktop. The dll does in fact validate this. So Tester-1, with TestExplorer on his desktop and RBT, can just ask Tester-2, who only has RBT on his desktop, to dump test cases into TestExplorer. Tester-2's RBT would simply inform him that TestExplorer was not installed. RBT and Test Explorer are using COM to communicate. When Test Explorer is installed, the COM server is registered with Windows, and after that, Windows knows where to find it.

Selecting **Utilities → Export to TestExplorer** brings up the following dialog:

Export to TestExplorer Dialog

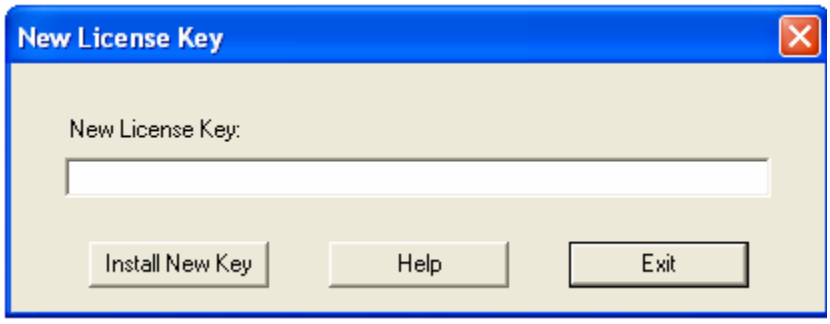
Once the tests have been exported, you can now find them in TestExplorer. Opening up the project and selecting one of the tests will show:



Tests in TestExplorer

12. New License Key

This option allows you to enter a new key for RBT. This would mainly be used in the case where the user had an evaluation key that was being extended or converted to a permanent key.



New License Key